Chapter 1

Software Engineering
Concerned with theories, methods and tools for professional software development

Software Development Cost Percentages
60% Development / 40% Testing

Two main causes of software project failure
1) Increasing system complexity
2)Failure to use software engineering methods

Attributes of good software
1) Maintainability
2) Dependability & Security
3) Efficiency
4) Usability (Acceptability)

What are the fundamental software engineering activities?
1) Specification
2) Development
3) Validation
4) Evolution

Difference between software engineering and computer science?
CS focuses on theory and fundamentals, software engineering is concerned with practicalities of developing and delivering useful software.

Key challenges facing software engineering?
Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.

Software Products: Generic
Stand-alone systems that are marketed and sold to any customer who wishes to buy them. (PC software such as graphics programs, project management tools, CAD software)
Software Products: Customized
Software that is commissioned by a specific customer to meet their own needs. (Embedded control systems, Air Traffic Control systems, traffic monitoring systems)

T or F: Software engineering is an ENGINEERING DISCIPLINE that is concerned with ALL ASPECTS of software production?
T: It uses appropriate theories and methods to solve problems and is not just concerned with the technical process of development but also project management and development of tools, methods etc

Why is software engineering important?
1) More and more individuals and society rely on advanced software systems. Its important to produce reliable and trustworthy systems economically and quickly.
2) It is usually cheaper in the long run to use SE methods and techniques.

General issues that affect software during development
1) Heterogeneity (systems that are required to operate across networks that include different types of computers and mobile devices)
2) Business and social change
3) Security and trust
4) Scale

Web Software engineering fundamentals
1) Software Reuse
2) Incremental and agile development
3) Service-oriented systems
4) Rich interfaces

Issues of professional responsibility
1) Confidentiality
2) Competence
3) Intellectual property rights
4) Computer misuse

ACM/IEEE Code of Ethics
The Code contains eight principles related to the behavior of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

Eight Principles of the ACM/IEEE Code of Ethics
1) Public
2) Client and Employer
3) Product
4) Judgment
5) Management
6) Profession
7) Colleagues
8) Self

The Three Software Process Models are:
- Waterfall model
- Incremental development
- Integration and configuration

What are the 8 different application types:
- Stand alone systems
- Interactive transaction-based applications
- Embedded control systems
- Batch processing systems
- Entertainment systems
- Data collection and analysis systems
- Systems of systems

_____ is an engineering discipline concerned with all aspects of software production.
Software engineering

Software is not just a program or programs but also includes all electronic _____ that is needed by systems users, QA staff, and developers.
documentation

Essential software product attributes are maintainability, _____, _____, and acceptability.
dependability and security
efficiency

The software process includes all of the activities involved in software development. The high-level activities of specification, _____, validation, and _____ are part of all software processes.
development
evolution

The fundamental ideas of software engineering are applicable to all types of software systems. These fundamentals include managed software processes, software dependability and security, _____ engineering, and software reuse.
requirements

CHAPTER 2

Software Process
A structured set of activities required to develop a software system.

Four main software processes
1) Specification
2) Design and Implementation
3) Validation
4) Evolution

Software Process Model
An abstract representation of a process. It presents a description of a process from some particular perspective.
When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities. Process descriptions may also include:
Products: The outcomes of a process activity
Roles: Reflect the responsibilities of the people involved
Pre- and Post-Conditions: Statements that are true before and after a process activity has been enacted or product produced

Describe the planning of Plan-driven processes
All of the process activities are planned in advance and progress is measured against this plan.

T or F: Most practical processes include elements of both plan-driven and agile approaches.
True!

Waterfall Model Process Model
Plan-driven model. Separate and distinct phases of specification and development.

Incremental Development Process Model
Specification, development and validation are interleaved. May be plan-driven or agile.

Integration and Configuration Process Model
The system is assembled from existing configurable components. May be plan-driven or agile.

Phases of the Waterfall model
1) Requirements definition
2) Systems and software design
3) Implementation and unit testing

4) Integration and system testing
5) Operation and maintenance

T or F: 80% of Waterfall type projects fail
True! (According to Prof. Fazi)

What is one of the main difficulties facing a pure waterfall model development project?
Accommodating change after the process is underway.

Systems where the waterfall method is more appropriate
1) Embedded systems where the software has to interface with hardware systems
2) Critical systems where there is a need for extensive safety and security analysis
3) Large software systems that are a part of broader engineering systems

Three main benefits of incremental development
1) The cost of accommodating changing customer requirements is reduced
2) It is easier to get customer feedback on the development work that has been done.
3) More rapid delivery and deployment of useful software to the customer

Two main concerns with incremental development
1) The process is not visible
2) System structure tends to degrade as new increments are added

The integration and configuration software process model is based on what major concept?
Software reuse: Reused elements may be configured to adapt their behavior and functionality to a user's requirements

T or F: Reuse is now the standard approach for building many types of business systems?
True!

Types of reusable software
1) Standalone application systems that are configured for use in a particular environment
2) Collections of objects that are developed as a package to be integrated into a framework
3) Web services that are developed according to service standards and which are available for remote invocation

Key process stages of reuse based software engineering
1) Requirements specification
2) Software discovery and evaluation
3) Requirements refinement
4) Application system configuration
5) Component adaptation and integration

Reuse-Based Development: Application system configuration stage
If an off-the-shelf application system is available that meets the requirements, then it is configured for use in the new system

Reuse-Based Development: Component adaptation and integration stage
If there is no off-the-shelf application available, individual reusable components may be adapted and new components developed. These are then integrated into the system

Advantages of reuse oriented software development
1) Reduced cost and risks as less software is developed from scratch
2) Faster delivery and deployment of system

Disadvantages of reuse oriented software development
1) Requirements compromises are inevitable so system may not meet real needs of users
2) Loss of control over evolution of reused system elements

Four basic process activities of software development
1) Specification
2) Development
3) Validation
4) Evolution

T or F: The four process activities of software development are organized the same regardless of what process model is being used
False! For example, in the waterfall model they are organized in sequence whereas in incremental development they are interleaved

Software Specification
The process of establishing what services are required and the constraints on the system's operation and development.

Three stages of the requirements engineering process
1) Requirements elicitation and analysis: What do the stakeholders require or expect?
2) Requirements specification: Defining those requirements in detail
3) Requirements validation: Checking the validity of those requirements

Software design and implementation
The process of converting the system specification into an executable system.

Four main design activities
1) Architectural design
2) Database design
3) Interface design
4) Component selection and design

Architectural design
Where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.

Database design
Where you design the system data structures and how these are to be represented in a database.

Interface design
Where you define the interfaces between system components.

Component selection and design
Where you search for reusable components. If unavailable, you design how it will operate.

Software validation (V & V)
Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer. Involves checking and review processes and system testing.

Three Stages of Testing
1) Component testing
2) System testing
3) Acceptance (Customer) testing

Component testing
The testing of individual software components.

System testing
Testing the entire system as one entity to ensure that it is working properly

Acceptance or Customer testing
Testing with customer data to check that the system meets the customer's needs.

Software evolution
Modifying or changing the software to meet changing customer needs.

Name some examples of how and why system requirements can change
External pressures (Usually means money)
Competition
Changed management priorities

Two related approaches to reducing the cost of rework
1) Change anticipation: activities that can anticipate possible changes before significant rework is required
2) Change tolerance: where the process is designed so that changes can be accommodated at relatively low cost

Coping with change: System Prototyping
Version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This approach supports change anticipation.

Coping with change: Incremental delivery
Where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance.

Benefits of Prototyping
1. Improved system usability
2. A closer match to users' real needs
3. Improved design quality
4. Improved maintainability
5. Reduced development effort

T or F: Prototypes should be discarded after development and why?
True! They are not a good basis for a production system because it may be impossible to tune the system to meet non-functional requirements, they are normally undocumented, the structure is degraded through rapid change, and usually they do not meet normal organizational quality standards

Incremental delivery advantages
1. Customer value can be delivered with each increment so system functionality is available earlier.
2. Early increments act as a prototype to help elicit requirements for later increments.
3. Lower risk of overall project failure.
4. The highest priority system services tend to receive the most testing.

Incremental delivery problems
1) Most systems require a set of basic facilities that are used by different parts of the system
2) The essence of iterative processes is that the specification is developed in conjunction with the software

Process Improvement
Understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

Two approaches to process improvement
1) Process maturity approach
2) The agile approach

Process improvement: Process maturity approach
Focuses on improving process and project management and introducing good software engineering practice. The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes

Process improvement: The agile approach
Focuses on iterative development and the reduction of overheads in the software process.

Process improvement activities
1) Measurement
2) Analysis
3) Change

Process maturity levels
1) Initial - Few processes defined
2) Repeatable - Basic project management, requirements, and documentation are known
3) Defined - Process management procedures and strategies defined and used
4) Managed - Quality management strategies defined and used
5) Optimizing - Process improvement strategies defined and used

CHAPTER 3

Agile methods are incremental development methods that focus on_____ , _____, _____ by minimizing documentation and producing _____ code.
rapid software development
frequent releases of the software
reducing process overheads
high-quality

Scrum is an _____ that provides a project management framework.
agile method

Many practical development methods are a mixture of _____and _____.
plan-based
agile development

Scrum is centered around a set of _____, which are fixed time periods when a system increment is developed.
sprints

Most important requirement for software systems
Rapid development and delivery

Common characteristics of agile development
- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation - focus on working code

Plan-driven development
Based around separate development stages with the outputs to be produced at each of these stages planned in advance.

Agile Development
Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

Five principles of Agile Methods
1) Customer involvement
2) Incremental delivery
3) People not process
4) Embrace change
5) Maintain simplicity

Agile Manifesto Tenets
- Individuals and Interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile methods are useful for what two kinds of system development?
1) Developing a small or medium-sized product for sale
2) Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are few external rules and regulations that affect the software

T or F: A "customer" must be from an external source outside the company.
False! A "customer" can be an organization from within the company as well.

Extreme Programming
A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.

Why is it called Extreme Programming?
- New versions may be built several times per day;
- Increments are delivered to customers every 2 weeks;
- All tests must be run for every build and the build is only accepted if tests run successfully.

Practices of Extreme Programming
1) Incremental planning
2) Small releases
3) Simple design
4) Test-first development
5) Refactoring
6) Pair programming
7) Collective ownership
8) Continuous integration
9) Sustainable pace
10) On-site customer

XP applied to the Agile Manifesto
1) Incremental development is supported through small, frequent system releases.
2) Customer involvement means full-time customer engagement with the team.
3) People not process through pair programming, collective ownership and a process that avoids long working hours.
4) Change supported through regular system releases.
5) Maintaining simplicity through constant refactoring of code.

Practical and influential XP practices
1) User stories for specification
2) Refactoring
3) Test-first development
4) Pair programming

User Stories
- User requirements are expressed as user stories or scenarios.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

Refactoring
- Constant code improvement
- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.

Examples of refactoring
- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

Test-first development
- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.

Extreme Programming Testing Features
1) Test-first development
2) Incremental test development from scenarios
3) User involvement in test development and validation
4) Automated test harnesses are used to run all component tests each time that a new release is built.

Problems with a test-first environment
1) Programmers prefer programming to testing and sometimes they take short cuts when writing tests. (debatable!)
2) Some tests can be very difficult to write incrementally.
3) It difficult to judge the completeness of a set of tests.

Pair Programming
- Pair programming involves programmers working in pairs, developing code together.

Benefits of Pair Programming
- Helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from improving the system code.

Scrum
Agile method that focuses on managing iterative development rather than specific agile practices.

Three phases of Scrum
1) The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
2) This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
3) The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

AAR
After Action Review (wrap-up)

Product Backlog
The "to-do" items which the Scrum team must tackle.

Scrum
A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Short, face-to-face meeting that includes the whole team.

ScrumMaster
Person responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company.
Sprint

A development iteration. Sprints are usually 2-4 weeks long.

Velocity
An estimate of how much product backlog effort that a team can cover in a single sprint.

Benefits of Scrum
1) The product is broken down into a set of manageable and understandable chunks.
2) Unstable requirements do not hold up progress.
3) The whole team have visibility of everything and consequently team communication is improved.
4) Customers see on-time delivery of increments and gain feedback on how the product works.
5) Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

"Scaling up" agile methods
Concerned with using agile methods for developing large software systems that cannot be developed by a small team.

"Scaling out" agile methods
Concerned with how agile methods can be introduced across a large organization with many years of software development experience.

Important agile fundamentals to remember when scaling
1) Flexible planning
2) Frequent system releases
3) Continuous integration
4) Test-driven development
5) Good team communications.

Practical problems with Agile Methods
1) The informality of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies.
2) Agile methods are most appropriate for new software development rather than software maintenance. Yet the majority of software costs in large companies come from maintaining their existing software systems.
3) Agile methods are designed for small co-located teams yet much software development now involves worldwide distributed teams.

Agile methods and software maintenance concerns
1) Lack of product documentation
2) Keeping customers involved
3) Development team continuity

Questions to be asked when considering using agile methods

- How large is the system being developed? (Agile methods are most effective a relatively small co-located team who can communicate informally.)
- What type of system is being developed?
(Systems that require a lot of analysis before implementation need a fairly detailed design to carry out this analysis.)
- What is the expected system lifetime?
(Long-lifetime systems require documentation to communicate the intentions of the system developers to the support team.)
- Is the system subject to external regulation?
(If a system is regulated you will probably be required to produce detailed documentation as part of the system safety case.)

T or F and Why?: Scaling agile methods for large systems is easy.
False! Large systems need up-front design and some documentation and organizational practice may conflict with the informality of agile approaches.

CHAPTER 4

Two types of requirements
User and system
User requirements

Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

system requirements
A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

System Stakeholders
Any person or organization who is affected by the system in some way and so who has a legitimate interest

Types of stakeholders
- End users
- System managers
- System owners
- External stakeholders

Functional Requirements
- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- May state what the system should not do.

Non-Functional Requirements
-Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
-Often apply to the system as a whole rather than individual features or services.

Domain requirements
Constraints on the system from the domain of operation rather than the from the specific needs of the system users

Requirements imprecision
• Problems arise when requirements are not precisely stated.
• Ambiguous requirements may be interpreted in different ways by developers and users.

• Consider the term 'search'
- User intention - search for a patient name across all appointments in all clinics;
- Developer interpretation - search for a patient name in an individual clinic. User chooses clinic then search.

Types of non-functional requirements
1) Product: Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
2) Organisational: Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
3) External: Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc

Metrics for specifying non-functional requirements
- Speed
- Size
- Ease of use
- Reliability
- Robustness
- Portability

Requirements Engineering process activities
1) Requirements elicitation
2) Requirements analysis
3) Requirements validation
4) Requirements management

Requirements elicitation and analysis
-Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.

Stages of Requirements Elicitation
1) Requirements discovery
2) Requirements classification and organization
3) Requirements prioritization and negotiation
4) Requirements specification

Difficulties of Requirements Elicitation
- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.

Two fundamental approaches to requirements elicitation
1) Interviewing
2) Observation or ethnography

Types of Interviews
1) Closed: interviews based on pre-determined list of questions
2) Open: where various issues are explored with stakeholders.

Problems with Interviews
- Application specialists may use language to describe their work that isn't easy for the requirements engineer to understand.
- Interviews are not good for understanding domain requirements

Keys to effective interviewing
- Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
- Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

Ethnography
-A social scientist spends a considerable time observing and analyzing how people actually work.
-People do not have to explain or articulate their work.
-Social and organisational factors of importance may be observed.

Pitfall of Ethnography
Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

Stories and Scenarios
-Scenarios and user stories are real-life examples of how a system can be used.
-Stories and scenarios are a description of how a system may be used for a particular task.
-Because they are based on a practical situation, stakeholders can relate to them and can comment on their situation with respect to the story.

A scenario should include:
1) A description of the starting situation
2) A description of the normal flow of events
3) A description of what can go wrong
5) Information about other concurrent activities
6) A description of the state when the scenario finishes

Requirements specification

- The process of writing down the user and system requirements in a requirements document.
- User requirements have to be understandable by end-users and customers who do not have a technical background.
- System requirements are more detailed requirements and may include more technical information.

Ways of writing a system requirements specification
-Natural language
-Structured and natural language
- Design description languages
- Graphical notations
- Mathematical specifications

Natural Language Specification
- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.

Guidelines for writing requirements
1) Invent a standard format and use it for all requirements.
2) Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
3) Use text highlighting to identify key parts of the requirement.
4) Avoid the use of computer jargon.
5) Include an explanation (rationale) of why a
requirement is necessary.

Problems with natural language
1) Lack of clarity - Precision is difficult without making the document difficult to read.
2) Requirements Confusion - Functional and non-functional requirements tend to be mixed up
3) Requirements amalgamation - Several different requirements may be expressed together.

Structured language specifications
An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way. (works well for say embedded systems but is too rigid for business system requirements)

Form-Based Specifications
• Definition of the function or entity.
• Description of inputs and where they come from.
• Description of outputs and where they go to.
• Information about the information needed for the computation and other entities used.
• Description of the action to be taken.

• Pre and post conditions (if appropriate).
• The side effects (if any) of the function.

Tabular Specification
- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.
- For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

Use Cases
• Use-cases are a kind of scenario that are included in the UML.
• Use cases identify the actors in an interaction and which describe the interaction itself.
• A set of use cases should describe all possible interactions with the system.
• High-level graphical model supplemented by more detailed tabular description
• UML sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.

The Software Requirements document
- Official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

T or F: Systems developed incrementally will, typically, have more detail in the requirements document.
FALSE, they will typically have less detail

Structure (sections) of a Requirements Document
- Preface
- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

Requirements Validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important. (Fixing a requirements error after delivery may cost u to 100 times the cost of fixing an implementation error)

Requirements Checking Types
1) Validity - Does the system provide the functions which best support the customer's needs?
2) Consistency - Are there any requirements conflicts?
3) Completeness - Are all functions required by the customer included?
4) Realism - Can the requirements be implemented given available budget and technology?
5) Verifiability - Can the requirements be checked?

Techniques for Requirements Validation
1) Requirements reviews - Systematic manual analysis of the requirements
2) Prototyping - Using an executable model of the system to check requirements.
3) Test-case generation - Developing tests for requirements to check test-ability

Requirements Reviews
• Regular reviews should be held while the requirements definition is being formulated.
• Both client and contractor staff should be involved in reviews.
• Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

Review Checks
1) Verifiability - Is the requirement realistically testable?
2) Comprehensibility - Is the requirement properly understood?
3) Traceability - Is the origin of the requirement clearly stated?
4) Adaptability - Can the requirement be changed without a large impact on other requirements?

Changing requirements
- The business and technical environment of the system always changes after installation. (New hardware, new legislation, changing business priorities etc.)
- The people who pay for a system and the users of that system are rarely the same people.
- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.

Requirements Management
The process of managing changing requirements during the requirements engineering process and system development.

Requirements management decisions

1) Requirements identification - Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
2) A change management process - This is the set of activities that assess the impact and cost of changes.
3) Traceability policies - These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
4) Tool support - Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Three principle stages to a change management process
1) Problem analysis and change specification
2) Change analysis and costing
3) Change implementation

_____ for a software system set out what the system should do and define _____ on its operation and implementation.
Requirements
constraints

_____requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
Functional

_____ requirements often constrain the system being developed and the development process being used.
Non-functional

The requirements engineering process is an iterative process that includes requirements _____, _____ and _____.
elicitation
specification
validation

Requirements elicitation is an iterative process that can be represented as a spiral of activities - requirements _____, requirements classification and organization, requirements _____ and requirements documentation.
discovery
negotiation
documentation

You can use a range of techniques for requirements elicitation including _____and ethnography. User stories and _____ may be used to facilitate discussions.
interviews
scenarios

Requirements _____ is the process of formally documenting the user and system requirements and creating a software requirements _____.
specification
document

Requirements _____ is the process of checking the requirements for validity, consistency, completeness, realism and verifiability.
validation
Business, organizational and technical changes inevitably lead to changes to the requirements for a software system. Requirements _____ is the process of managing and controlling these changes.
management

CHAPTER 5

System Modeling
The process of developing abstract models of a system, with each model presenting a different view or perspective of that system

Universal Modeling Language
UML is a set of 13 different diagram types that may be used to model software systems. Emerged from work in the 1990s on object-oriented modeling, where similar object-oriented notations were integrated to create the UML.

T or F: A model is meant to be a complete representation of a system.
False, it purposely leaves out detail to make it easier to understand.

System Perspectives
1) An external perspective: where you model the context or environment of the system.
2) An interaction perspective: where you model the interactions between a system and its environment, or between the components of a system.
3) A structural perspective: where you model the organization of a system or the structure of the data that is processed by the system.
4) A behavioral perspective: where you model the dynamic behavior of the system and how it responds to events.

Three ways in which graphical models are used
1) As a means of facilitating discussion about an existing or proposed system
2) As a way of documenting an existing system
3) As a detailed system description that can be used to generate a system implementation

UML Diagram Types
~Activity diagrams
~Use case diagrams
~Sequence diagrams
~Class diagrams
~State diagrams

Activity Diagrams
Show the activities involved in a process or in data processing

Use Case Diagrams
Show the interactions between a system and its environment.

Sequence Diagrams
Show interactions between actors and the system and between system components.

Class Diagrams
Show the object classes in the system and the associations between these classes.

State Diagrams
Show how the system reacts to internal and external events.

Context Models
Used to illustrate the operational context of a system - they show what lies outside the system boundaries.

System Boundaries
- Established to define what is inside and what is outside the system.
- They show other systems that are used or depend on the system being developed.

What is meant when saying that defining a system boundary is a political judgment?
There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.

What two diagrams are used for interaction modeling?
- Use Case Modeling
- Sequence Diagrams

Use Case cases represent what?
A discrete task that involves external interaction with a system

What do sequence diagrams show?
The sequence of interactions that take place during a particular use case or use case instance.

What do structural models display?
The organization of a system in terms of the components that make up that system and their relationships.

Difference between static and dynamic structural models
- Static: Show the structure of the system design.
- Dynamic: Show the organization of the system when it is executing.

Class Diagram
An object-oriented system model to show the classes in a system and the associations between these classes.

T or F: When developing a class diagram, objects represent something in the real world.
True!

Generalization
- An everyday technique that we use to manage complexity.
- Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes.

How is generalization implemented in OO langauges?
- class inheritance mechanisms
- the attributes and operations associated with higher-level classes are also associated with the lower-level classes.
- The lower-level classes are subclasses inherit the attributes and operations from their superclasses. These lower-level classes then add more specific attributes and operations.

What does an aggregation model show?
- How classes that are collections are composed of other classes.
- Aggregation models are similar to the part-of relationship in semantic data models.
Behavior Models

Models of the dynamic behavior of a system as it is executing.

Two types of behavior when discussing Behavior Models
1) Data: Some data arrives that has to be processed by the system
2) Events: Some event happens that triggers system processing. Events may have associated data, although this is not always the case.

Data-Driven Models
- Show the sequence of actions involved in processing input data and generating an associated output.

- Particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system.

Event-Driven Models
Shows how a system responds to external and internal events

State Models
- Model the behaviour of the system in response to external and internal events.

- Show the system's responses to stimuli so are often used for modelling real-time systems.

Model-Driven Engineering
An approach to software development whereby models rather than programs are the principle outputs of the development process.

What is the main argument for proponents of MDE
They argue that this raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms

Pros of Model Driven Engineering
- Allows systems to be considered at higher levels of abstraction

- Generating code automatically means that it is cheaper to adapt systems to new platforms.
Cons of Model Driven Engineering
- Models for abstraction and not necessarily right for implementation.

- Savings from generating code may be outweighed by the costs of developing translators for new platforms

Model Driven Architecture
- precursor of more general model-driven engineering

-model-focused approach to software design and implementation that uses a subset of UML models to describe a system.

Types of MDA Models
1) Computational Independent Model (CIM)

2) Platform Independent Model (PIM)

3) Platform Specific Model (PSM)

Computation independent model (CIM)
These model the important domain abstractions used in a system. CIMs are sometimes called domain models.

Platform independent model (PIM)
These model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.

Platform specific models (PSM)
These are transformations of the platform-independent model with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.

Reasons why MDA/MDE has not become more mainstream
- Specialized tool support is required to convert models from one level to another

- There is limited tool availability and organizations may require tool adaptation and customisation to their environment

- For the long-lifetime systems developed using MDA, companies are reluctant to develop their own tools or rely on small companies that may go out of business

- Models are a good way of facilitating discussions about a software design. Howeverthe abstractions that are useful for discussions may not be the right abstractions for implementation.

- For most complex systems, implementation is not the major problem - requirements engineering, security and dependability, integration with legacy systems and testing are all more significant.

A model is an _____ view of a system that ignores system details. Complementary system models can be developed to show the system's _____, interactions, structure and _____.
abstract

context

behavior

_____ models show how a system that is being modeled is positioned in an environment with other systems and processes.
Context

Use case diagrams and sequence diagrams are used to describe the interactions between _____ and _____ in the system being designed. _____ describe interactions between a system and external actors; _____add more information to these by showing interactions between system objects.
users

systems

Use cases

sequence diagrams

_____ models show the organization and architecture of a system. _____are used to define the static structure of classes in a system and their associations.
Structural

Class diagrams
Types of diagrams used for interaction models
- Use Cases

- Sequence Diagrams
Types of diagrams used for structural models
- Class Diagrams

Types of diagrams used for behavioral models
- Activity Diagrams

- Sequence Diagrams

Types of diagrams used for context models
Activity diagrams

_____models are used to describe the dynamic behavior of an executing system. This behavior can be modeled from the perspective of the data processed by the system, or by the events that stimulate responses from a system.
Behavioral

_____ diagrams may be used to model the processing of data, where each activity represents one process step.
Activity

_____diagrams are used to model a system's behavior in response to internal or external events.
State

_____ is an approach to software development in which a system is represented as a set of models that can be automatically transformed to executable code.
Model-driven engineering

When are system models used?
During the requirements engineering phase to help explain the proposed requirements to other stakeholders.

CHAPTER 6

Architectural Design
- Understanding how a software system should be organized and designing the overall structure of that system. First stage in the software design process.

- Critical link between design and requirements engineering.

- Output is an architectural model that describes how the system is organized as a set of communicating components.

Two levels of abstraction in software architecture
- Architecture in the small
- Architecture in the large

Architecture in the small
Concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components.

Architecture in the large
Concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies

Main advantages of designing and documenting software architecture
1. Stakeholder communication
2. System Analysis
3. Large Scale Resuse

T or F: Individual components implement the functional system requirements, but the dominant influence on the non-functional system characteristics is the system's architecture.
True!

What are block diagrams useful for?
- People from different disciplines can easily understand the scope of a project.

- Useful for communication with stakeholders.
What are the downsides to simple, informal block diagrams?

- They lack semantics

- They do not show the different types of relationships between entities, nor their properties

Two main uses of architectural models
- As a way of facilitating discussion about the system design (High level view of a system is useful for communication with system stakeholders and project planning)

- As a way of documenting an architecture that has been designed

What are some Architectural Design decisions?
- Is there a generic application that can act as a template?

- How will it be distributed?

- What architectural patterns or styles might be used?

- What strategy will control the operation of the components in the system?

- How should the architecture of the system be documented?

- What arch. organization is best for delivering the non-functional requirements?

Non-Functional requirements that the choice of architectural style and structure should depend on:
1) Performance - localize critical operations

2) Security - Use a layered architecture with critical assets in an inner layer.

3) Safety - Localize safety-critical features in a small number of subsystems.

4) Availability - Include redundant components and mechanisms for fault tolerance.

5) Maintainability - Use fine-grain, replaceable components.

Architecture Reuse
- Systems in the same domain often have similar architectures that reflect domain concepts.*

- Application product lines are built around a core architecture with variants that satisfy particular customer requirements.

- The architecture of a system may be designed around one of more architectural patterns or 'styles'.

- These capture the essence of an architecture and can be instantiated in different ways.

The 4+1 Views of Software Architecture
- Logical View
- Process View
- Development View
- Physical View
- Related use cases or scenarios (+1)

Logical View
Shows the key abstractions in the system as objects or object classes.

Process View
Shows how, at run-time, the system is composed of interacting processes.

Development View
Shows how the software is decomposed for development

Physical View
Shows the system hardware and how software components are distributed across the processors in the system

Architectural Patterns
- Means of representing, sharing, and reusing knowledge.

- Good design practice

- Patterns should include information about when they are and when they are not useful

- Can be represented using tabular or graphical descriptions

The Model-View-Controller (MVC) Pattern

- Separates presentation and interaction from the system data

- Model component manages the system data and associated operations

- View component defines and manages how the data is presented to the user

- Controller component manages user interaction and passes these interactions to the view and model components.

When should MVC patterns be used?
When there are multiple ways to view and interact with data. Also when the future requirements for interaction and presentation of data are unknown.

Model-View-Controller: Advantages
- Allows the data to change independently of its representation and vice versa.

- Supports presentation of the same data in different ways with changes made in one representation shown in all of them

Model-View-Controller: Disadvantages
- Can involve additional code and code complexity when the data model and interactions are simple
Layered Architectural Pattern
- Organizes the system into layers wuith related functionality associated with each layer.

- A layer provides services to the layer above it.

- Services of the lowest layer are then used throughout the system.

When should a layered architectural pattern be employed?
- When building new facilities on top of existing systems

- When development is spread across several teams with each team responsible for a layer

- When there is a requirement for multi-level security

Layered Pattern: Advantages

- Allows for replacement of entire layers so long as the interface is maintained.

- Redundant facilities can be provided in each layer to increase the dependability of the system.

Layered Pattern: Disadvantages
- In practice, providing clean separation between layers in often difficult

- Performance can be a concern because of the multiple levels of interpretation of a service request as it is processed in each layer.

Repository Architectural Pattern
- All data in a system is managed in a central repository that is accessible to all system components

- Components do not interact directly, only through the repository

When should you use a repository pattern?
- When you have a system in which large volumes of information are generated that has to be stored for a long time.

Repository Pattern: Advantages
- Components can be independent, they do not need to know of the existence of other components
- Changes made by one component can be propagated to all components.

- All data can be managed consistently!

Repository Pattern: Disadvantages
- The repository is a single point of failure so problems in the repository can affect the whole system.

- May be inefficiencies in organizing all communication through the repository.

- Distributing the repository across several computers may be difficult.

The Client-Server Pattern
- The functionality of the system is organized into servers, with each service delivered from a separate server.

- Clients are users of these services and access servers to make use of them.

When should you use a Client-Server Pattern?
- When data in a shared database has to be accessed from a range of locations.

Client-Server Pattern: Advantages
- Servers can be distributed across a network

- General functionality (e.g. a printing service) can be available to all clients and does not need to be implemented by all services.

Client-Server Pattern: Disadvantages
- Each service is a single point of failure, susceptible to denial of service attacks or server failure.

- Performance can be unpredictable because it depends on both the network and they system.

Pipe and Filter Architectural Pattern
- Processing of data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation.

- The data flows (pipe) from one component to another for processing.

When should a Pipe and Filter pattern be used?
- Commonly used in data processing applications where inputs are process in separate stages to generate related outputs.

Pipe and Filter Pattern: Advantages
- Easy to understand and supports transformation reuse

- Workflow style matches the structure of many business processes.

- Evolution by adding transformations is straight forward.

- Can be implemented as either a sequential or concurrent system

Pipe and Filter Pattern: Disadvantages
- Format for data transfer has to be agreed upon between communicating transformations.

- Each transformation must parse its input and un-parse its output to an agreed upon form. This increases system overhead and may mean that it's impossible to reuse functional transformations that use incompatible data structures.

Application Architectures
- Designed to meet an organizational need

- Generic application architecture is for a type of system to be configured and adapted to meet specific requirements.

Uses for models of application architecture
- As a starting point for architectural design

- As a design checklist

- As a way of organizing the work of a development team

- As a means of assessing components for reuse

- As a vocabulary for talking about application types

Examples of Application types
-Data Processing Apps

- Transaction Processing Apps

- Event Processing Apps

- Language Processing Apps

Two most widely used generic application architectures
- Transaction Processing systems: Ecommerce, reservation systems

- Language Processing Systems: Compiler, command interpreters

A software's _____ is a description of how a software system is organized.
architecture

Examples of Architectural design _____include the type of application, the distribution of the system, and the architectural styles to be used.
decisions

Architectures may be documented from several different perspectives or views such as a _____view, a logical view, a _____view, and a development view.
conceptual
process

Architectural patterns are a means of _____knowledge about generic system architectures. They describe the architecture, explain when it may be used and describe its advantages and disadvantages.
reusing

_____ processing systems are interactive systems that allow information in a database to be remotely accessed and modified by a number of users.
Transaction

_____processing systems are used to translate texts from one language into another and to carry out the instructions specified in the input language. They include a translator and an abstract machine that executes the generated language.
Language

CHAPTER 7

Software Design and Implementation
- The stage in the software engineering process at which an executable software system is developed.

- Software design and implementation activities are invariably inter-leaved.

- Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements.

- Implementation is the process of realizing the design as a program.

Object-Oriented Design Process
- Structured object-oriented design processes involve developing a number of different system models.

- They require a lot of effort for development and maintenance of these models and, for small systems, this may not be cost-effective.

- However, for large systems developed by different groups design models are an important communication mechanism.

Process stages of OO Design Process
- Define the the context and modes of use of the system

- Design the system architecture

- Identify the principle system objects

- Develop design models

- Specify object interfaces

System Context and Interactions
- Understanding the relationships between the software being designed and the external environment.

- Helps you to establish the boundaries of the system

Context and Interaction Models
- System context model is a structured model that demonstrates the other systems in the environment of the system being developed.

- Interaction is a dynamic model that shows how the system interacts with its environment as it is used.

Architectural Design
- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.

- Identify the major components that make up the system and their interactions, and then organize the components using an architectural pattern such as a layered or client-server model.

Object Class Indentifcation
- Often difficult part of object orientated design

- No magic formula! Relies on skill, experience, and domain knowledge of system designers

- Object identification is an iterative process, unlikely to get it right the first time!

Approaches to Object Class Identification
- Use a grammatical approach based on a natural language description of the system.

- Base the identification on tangible things in the application domain.

- Use a behavioral approach and identify objects based on what participates in what behavior.

- Use a scenario-based analysis. The objects, attributes and methods in each scenario are identified.

Design Models
- Show the objects and object classes and relationships between these entities

Two kinds of Design Models
- Structural Models: Describe the static structure of system in terms of object classes and relationships

- Dynamic Models: Describe the dynamic interactions between objects

Examples of design models
- Subsystem models

- Sequence models

- State Machine Models

- Use-Case

- Aggregation

- generalization

Subsystem Models
Shows how the design is organised into logically related groups of objects

Sequence Models
- Shows sequence of object interactions that take place

- Objects are arranged horizontally across the top

- Time is represented vertically

State Diagrams
- Used to show how objects respond to different service requests and the state transitions triggered by these requests.

- State diagrams are useful high-level models of a system or an object's run-time behavior.

- Don't need a state diagram for all objects in the system

Interface Specification
- Object interfaces have to be specified so that the objects and other components can be designed in parallel.

- Designers should avoid designing the interface representation but should hide this in the

object itself.

- Objects may have several interfaces which are viewpoints on the methods provided.

- The UML uses class diagrams for interface specification but Java may also be used.

Design Patterns
- A design pattern is a way of reusing abstract knowledge about a problem and its solution.

- A pattern is a description of the problem and the essence of its solution.

- It should be sufficiently abstract to be reused in different settings.

- Pattern descriptions usually make use of object-oriented characteristics such as inheritance and polymorphism.

Design Pattern Elements
1) Name: A meaningful pattern identifier

2) Problem Description

3) Solution Description: Not a concrete design by a template for a design solution

4) Consequences: The results and trade-offs of applying the pattern

Elements of the Observer Pattern
Name: Observer

Description: Separates the display of an object state from the object itself

Problem Description: Used when multiple displays of data are needed

Solution Description: See UML Description

Consequences: Optimizations to enhance display performance are impractical

Implementation Issues
- Reuse

- Configuration Management

- Host-Target Development

Reuse
- From the 1960s to the 1990s, most new software was developed from scratch, by writing all code in a high-level programming language.

- The only significant reuse or software was the reuse of functions and objects in programming language libraries.

- Costs and schedule pressure mean that this approach became increasingly unviable, especially for commercial and Internet-based systems.

- An approach to development based around the reuse of existing software emerged and is now generally used for business and scientific software.

Levels of Reuse
- The Abstraction Level

- The Object Level

- The Component Level

- The System Level

Reuse: The Abstraction Level
At this level, you don't reuse software directly but use knowledge of successful abstractions in the design of your software.

Reuse: The Object Level
At this level, you directly reuse objects from a library rather than writing the code yourself.

Reuse: The Component Level
Components are collections of objects and object classes that you reuse in application systems.

Reuse: The System Level
At this level, you reuse entire application systems.

Costs associated with reuse
- Time spent looking for software to reuse and assessing whether or not it meets your needs

- Costs of buying the reusable software

- Cost of adapting and configuring

- Cost of integrating the reusable software elements with each other

Configuration Management
- General process of managing a changing software system.

- Aim of configuration management is to support the system integration process so that all developers can access the project code and documents in a controlled way, find out what changes have been made, and compile and link components to create a system.

Configuration Management Activities
- Version Management

- System Integration

- Problem Tracking

Configuration Management: Version Management
Where support is provided to keep track of the different versions of software components. Version management systems include facilities to coordinate development by several programmers.

Configuration Management: System Integration
Where support is provided to help developers define what versions of components are used to create each version of a system. This description is then used to build a system automatically by compiling and linking the required components.

Configuration Management: Problem Tracking
Where support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed.

Host-Target Development
- Most software is developed on one computer (the host), but runs on a separate machine (the target).

- More generally, we can talk about a development platform and an execution platform.

- A platform is more than just hardware. It includes the installed operating system plus other supporting software such as a database management system

- Development platform usually has different installed software than execution platform; these platforms may have different architectures.

Development Platform Tools
- An integrated compiler and syntax-directed editing system

- A language debugging system

- Graphical editing tools

- Testing tools

- Project organizing tools

Integrated Development Environments (IDEs)
Set of software tools that supports different aspects of software development, within some common framework and user interface.

Component/System Deployment factors
- The hardware and software requirements of a component

- The availability of requirements of the system

- Component Communications

Open Source Development
- Source code of a software system is published and volunteers are invited to participate in the development process

- Roots are in the Free Software Foundation which advocates that source code should not be proprietary but rather should always be available for users to examine and modify as they wish.

- Internet helped to recruit a much larger population

Examples of open source software
Linux, Java, Apache web server, MySQL

Open Source Licensing
- A fundamental principle of open-source development is that source code should be freely available, this does not mean that anyone can do as they wish with that code.

- Legally, the developer of the code (either a company or an individual) still owns the code. They can place restrictions on how it is used by including legally binding conditions in an open source software license.

License Models
- GNU General Public License (GPL): So called reciprocal license. Means that if you use open source software licensed under the GPL, then you must make that software open source

- GNU Lesser General Public License (LGPL): a variant of the GPL license where you can write components that link to open source code without having to publish the source of these components.

- Berkley Standard Distribution (BSD) License: Non-Reciprocal license
License Management
- Establish a system for maintaining information about open-source components that are downloaded and used.

- Be aware of the different types of licenses and understand how a component is licensed before it is used.

- Be aware of evolution pathways for components.

- Educate people about open source.

- Have auditing systems in place.

- Participate in the open source community.

Software design and implementation are _____ activities. The level of detail in the design depends on the type of system and whether you are using a plan-driven or agile approach.
inter-leaved

The process of object-oriented design includes activities to _____ the system architecture, _____ objects in the system, _____ the design using different object models and _____ the component interfaces.
design
identify
describe
document

A range of different models may be produced during an object-oriented design process. These include _____ models (class models, generalization models, association models) and _____ models (sequence models, state machine models).
static
dynamic

When developing software, you should always consider the possibility of _____existing software, either as components, services or complete systems.
reusing

_____ management is the process of managing changes to an evolving software system. It is essential when a team of people are cooperating to develop software.
Configuration

Most software development is _____-target development. You use an IDE on a host machine to develop the software, which is transferred to a target machine for execution.
host

Open source development involves making the _____ of a system publicly available. This means that many people can propose changes and improvements to the software.
source code

CHAPTER 8

Elements of: Automated Testing
- Setup
- Call
- Assertion

Elements of: Development Testing
- Unit
- Component
- System

Elements of: Release Testing
- Requirements
- Scenario
- Performance

Elements of: Test Driven Development
- Identify
- Write
- Run
- Implement
- Repeat

Elements of: User Testing
- Alpha
- Beta
- Acceptance

Verification & Validation
- Verification: "Are we building the product right?"

- Validation: "Are we building the right product?"

Development Testing
- AKA Defect Testing, the system is tested to discover bugs and defects

- Includes all testing activities that are carried out by the team developing the system

- Testing activities include Unit testing, Component testing, and System testing

Unit Testing
- Stage of Development testing

- Process of testing individual components in isolation.

- It is a defect testing process

- Components to be tested could be individual functions or methods, object classes, or composite components with defined interfaces used to access their functionality.

Partition Testing
- Test case selection strategy for Unit testing

- Identify groups of inputs that have common characteristics and should be processed in the same way.

- Test cases are chosen from these groups

Guideline-based Testing
- Test case selection strategy for Unit testing

- A set of rules are used to choose test cases

- These rules reflect previous experience of the kinds or errors that programmers often make when developing components.

Automated Testing
- Recommended wherever possible for Unit Testing

- Making use of a test automation framework (such as JUnit) to write and run program tests.

- These frameworks provide generic test cases that you extend to create specific test cases.

Component Interface Testing
- Stage of Development testing

- Software components are often composite components that are made up of several interacting objects accessed through defined component interfaces.

- This type of testing should then focus on showing that the interface behaves according to its specification.

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces

System Testing
- Stage of Development testing

- Involves integrating components to create a version of the system and then testing the integrating system

- Focus is testing the interactions between components

- Checks to ensure that components are compatible, interact correctly, and transfer the right data at the right time.

- Might include stress testing

Test Driven Development (TTD)
- Approach to program development in which you inter-leave testing and code development

- Tests are written before coding and 'passing' the tests is the critical driver of development

- Introduced as a part of agile development methods like extreme programming but can also be employed in plan-driven development processes.

Regression Testing
- Benefit/Part of Test Driven Development

- A test suite is developed incrementally as a program is developed.

- Checking to ensure that changes to the code have not broken the previously working code.

Release Testing
- Process of testing a particular release of a system that is intended for use outside of the development team

- A separate testing team tests a complete version of the system

- Primary goal is to convince the supplier of the system that it is good enough for use.

- Usually a black-box testing process where tests are only derived from the system specification.

Requirements Based Testing
- Part of Release Testing

- Involves examining each requirement and developing a test or tests for it.

- Testing what was written up for and documented for the system.

- It is validation rather than defect testing: you are trying to demonstrate that the system has properly implemented its requirements.

Scenario Testing
- Part of release testing.

- An approach to release testing whereby you devise typical scenarios of use and use these scenarios to develop test cases for the system.

Performance Testing
- Part of release testing

- AKA Stress Testing

- Concerned with both demonstrating that the system meets its requirements and discovering problems and defects within the system.

- You are testing the system by making demands that are outside the design limitations.

User Testing
- Potential users of a system operate the system in their own environment.

Alpha Testing
- Stage of user testing

- Users of the software work with the development team to test the software at the developer's site.

Beta Testing
- Stage of user testing

- A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.

Acceptance Testing
- Stage of user testing

- Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

When testing software you are doing two things, what are they ?
1. Demonstrate to the developer and the customer that the software meets its requirements. For custom software, this means that there should be at least one test for every requirement in the requirements document. For generic software products, it means that there should be tests for all of the system features that will be included in the product release. You may also test combinations of features to check for unwanted interactions between them.

2. Find inputs or input sequences where the behavior of the software is incorrect, undesirable, or does not conform to its specification. These are caused by defects (bugs) in the software. When you test software to find defects, you are trying to root out undesirable system behavior such as system crashes, unwanted interac- tions with other systems, incorrect computations, and data corruption.

Validation question is ?
Are we building the right product?

Verification question is ?
Are we building the product right?

The level of required confidence depends on the sys- tem's purpose, the expectations of the system users, and the current marketing environment for the system
1. Software purpose The more critical the software, the more important it is that it is reliable. For example, the level of confidence required for software used to control a safety-critical system is much higher than that required for a demon- strator system that prototypes new product ideas.

2. User expectations Because of their previous experiences with buggy, unreliable software, users sometimes have low expectations of software quality. They are not surprised when their software fails. When a new system is installed, usersmay tolerate failures because the benefits of use outweigh the costs of failure recovery. However, as a software product becomes more established, users expect it to become more reliable. Consequently, more thorough testing of later versions of the system may be required.

3. Marketing environment When a software company brings a system to market, it must take into account competing products, the price that customers are willing to pay for a system, and the required schedule for delivering that system. In a competitive environment, the company may decide to release a program before it has been fully tested and debugged because it wants to be the first into the market. If a software product or app is very cheap, users may be willing to toler- ate a lower level of reliability.

Software inspection has three advantages over testing and are ?
1. During testing, errors can mask (hide) other errors. When an error leads to unexpected outputs, you can never be sure if later output anomalies are due to a new error or are side effects of the original error. Because inspection doesn't involve executing the system, you don't have to worry about interactions between errors. Consequently, a single inspection session can discover many errors in a system.

2. Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then you need to develop specialized test harnesses to test the parts that are available. This obviously adds to the system development costs.

3. As well as searching for program defects, an inspection can also consider broader quality attributes of a program, such as compliance with standards, portability, and maintainability. You can look for inefficiencies, inappropriate algorithms, and poor programming style that could make the system difficult to maintain and update.

Software system has to go through three stages of testing, what are they ?
1. Development testing, where the system is tested during development to discover bugs and defects. System designers and programmers are likely to be involved in the testing process.

2. Release testing, where a separate testing team tests a complete version of the system before it is released to users. The aim of release testing is to check that the system meets the requirements of the system stakeholders.

3. User testing, where users or potential users of a system test the system in their own environment. For software products, the "user" may be an internal market- ing group that decides if the software can be marketed, released and sold. Acceptance testing is one type of

user testing where the customer formally tests a system to decide if it should be accepted from the system supplier or if further development is required.

There are three stages of development testing, what are they ?
1. Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.

2. Component testing, where several individual units are integrated to create com- posite components. Component testing should focus on testing the component interfaces that provide access to the component functions.

3. System testing, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing com- ponent interactions.

Automated tests have three parts, what are they ?
1. A setup part, where you initialize the system with the test case, namely, the inputs and expected outputs.

2. A call part, where you call the object or method to be tested.

3. An assertion part, where you compare the result of the call with the expected result. If the assertion evaluates to true, the test has been successful; if false, then it has failed.

Effectiveness in testing means two things ?
1. The test cases should show that, when used as expected, the component that you are testing does what it is supposed to do.

2. If there are defects in the component, these should be revealed by test cases.

Two strtagegies that can bee effective in helping in test cases are ?
1. Partition testing, where you identify groups of inputs that have common charac- teristics and should be processed in the same way. You should choose tests from within each of these groups.

2. Guideline-based testing, where you use testing guidelines to choose test cases. These guidelines reflect previous experience of the kinds of errors that program- mers often make when developing components.

When testing programs what can help to reveal defects ?
1. Test software with sequences that have only a single value. Programmers natu- rally think of sequences as made up of several values, and sometimes they embed this assumption in their programs. Consequently, if presented with a single-value sequence, a program may not work properly.

2. Use different sequences of different sizes in different tests. This decreases the chances that a program with defects will accidentally produce a correct output because of some accidental characteristics of the input.

3. Derive tests so that the first, middle, and last elements of the sequence are accessed. This approach reveals problems at partition boundaries.

What are the different types of interface errors that can occur ?
1. Parameter interfaces These are interfaces in which data or sometimes function references are passed from one component to another. Methods in an object have a parameter interface.

2. Shared memory interfaces These are interfaces in which a block of memory is shared between components. Data is placed in the memory by one subsystem and retrieved from there by other subsystems. This type of interface is used in embedded systems, where sensors create data that is retrieved and processed by other system components.

3. Procedural interfaces These are interfaces in which one component encapsu- lates a set of procedures that can be called by other components. Objects and reusable components have this form of interface.

4. Message passing interfaces These are interfaces in which one component requests a service from another component by passing a message to it. A return message includes the results of executing the service. Some object-oriented sys- tems have this form of interface, as do client-server systems

Interface errors fall into three categories, what are they ?

■ Interface misuse A calling component calls some other component and makes an error in the use of its interface. This type of error is common in parameter inter- faces, where parameters may be of the wrong type or be passed in the wrong order, or the wrong number of parameters may be passed.

■ Interface misunderstanding A calling component misunderstands the specification of the interface of the called component and makes assumptions about its behavior. The called component does not behave as expected, which then causes unexpected behavior in the calling component. For example, a binary search method may be called with a parameter that is an

unordered array. The search would then fail.

■ Timing errors These occur in real-time systems that use a shared memory or a message-passing interface. The producer of data and the consumer of data may operate at different speeds. Unless particular care is taken in the interface design, the consumer can access out-of-date information because the producer of the information has not updated the shared interface information.

What are the general guidelines for interface testing ?
1. Examine the code to be tested and identify each call to an external component. Design a set of tests in which the values of the parameters to the external com- ponents are at the extreme ends of their ranges. These extreme values are most likely to reveal interface inconsistencies.

2. Where pointers are passed across an interface, always test the interface with null pointer parameters.

3. Where a component is called through a procedural interface, design tests that deliberately cause the component to fail. Differing failure assumptions are one of the most common specification misunderstandings.

4. Use stress testing in message passing systems. This means that you should design tests that generate many more messages than are likely to occur in prac- tice. This is an effective way of revealing timing problems.

5. Where several components interact through shared memory, design tests that vary the order in which these components are activated. These tests may reveal implicit assumptions made by the programmer about the order in which the shared data is produced and consumed.

In system testing the interface testing can overlap, what are the two different differences ?
1. During system testing, reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components. The complete system is then tested.

2. Components developed by different team members or subteams may be integrated at this stage. System testing is a collective rather than an individual process. In some companies, system testing may involve a separate testing team with no involvement from designers and programmers.

What inputs and outputs are required in sequence diagram ?
1. An input of a request for a report should have an associated acknowledgment. A report should ultimately be returned from the request. During testing, you should create summarized

data that can be used to check that the report is cor- rectly organized.

2. An input request for a report to WeatherStation results in a summarized report being generated. You can test this in isolation by creating raw data correspond- ing to the summary that you have prepared for the test of SatComms and check- ing that the WeatherStation object correctly produces this summary. This raw data is also used to test the WeatherData object.

What are the fundamental process in TDD or Test Driven Development ?
1. You start by identifying the increment of functionality that is required. This should normally be small and implementable in a few lines of code.

2. You write a test for this functionality and implement it as an automated test. This means that the test can be executed and will report whether or not it has passed or failed.

3. You then run the test, along with all other tests that have been implemented. Initially, you have not implemented the functionality so the new test will fail. This is deliberate as it shows that the test adds something to the test set.

4. You then implement the functionality and re-run the test. This may involve refactoring existing code to improve it and add new code to what's already there.

5. Once all tests run successfully, you move on to implementing the next chunk of functionality.

What are benefits of test-driven development ?
1. Code coverage In principle, every code segment that you write should have at least one associated test. Therefore, you can be confident that all of the code in the system has actually been executed. Code is tested as it is written, so defects are discovered early in the development process.

2. Regression testing A test suite is developed incrementally as a program is devel- oped. You can always run regression tests to check that changes to the program have not introduced new bugs.

3. Simplified debugging When a test fails, it should be obvious where the prob- lem lies. The newly written code needs to be checked and modified. You do not need to use debugging tools to locate the problem. Reports of the use of TDD suggest that it is hardly ever necessary to use an automated debugger in test-driven development (Martin 2007).

4. System documentation The tests themselves act as a form of documentation that describe what the code should be doing. Reading the tests can make it easier to understand the code.

There are two important distinctions between release testing and system testing during the development process, what are they ?
1. The system development, team should not be responsible for release testing.

2. Release testing is a process of validation checking to ensure that a system meets its requirements and is good enough for use by system customers. System test- ing by the development team should focus on discovering bugs in the system (defect testing).

Stress test helps you to do two things which are ?
1. Test the failure behavior of the system. Circumstances may arise through an unexpected combination of events where the load placed on the system exceeds the maximum anticipated load. In these circumstances, system failure should not cause data corruption or unexpected loss of user services. Stress testing checks that overloading the system causes it to "fail-soft" rather than collapse under its load.

2. Reveal defects that only show up when the system is fully loaded. Although it can be argued that these defects are unlikely to cause system failures in normal use, there may be unusual combinations of circumstances that the stress testing replicates.

There are three types of user testing, what are they ?
1. Alpha testing, where a selected group of software users work closely with the development team to test early releases of the software.

2. Beta testing, where a release of the software is made available to a larger group of users to allow them to experiment and to raise problems that they discover with the system developers.

3. Acceptance testing, where customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment.

Six stages of acceptance testing is ?
1. Define acceptance criteria This stage should ideally take place early in the pro- cess before the contract for the system is signed. The acceptance criteria should be part of the system contract and be approved by the customer and the devel- oper. In practice, however, it can be difficult to define criteria so early in the process. Detailed requirements may not be available, and the requirements will almost certainly change during the development process.

2. Plan acceptance testing This stage involves deciding on the resources, time, and budget for acceptance testing and establishing a testing schedule. The accept- ance test plan should also discuss the required coverage of the requirements and the order in which system features are tested. It should define risks to the testing process such as system crashes and inadequate performance, and discuss how these risks can be mitigated.

3. Derive acceptance tests Once acceptance criteria have been established, tests have to be designed to check whether or not a system is acceptable. Acceptance tests should aim to test both the functional and non-functional characteristics (e.g., performance) of the system. They should ideally provide complete cover- age of the system requirements. In practice, it is difficult to establish completely objective acceptance criteria. There is often scope for argument about whether or not a test shows that a criterion has definitely been met.

4. Run acceptance tests The agreed acceptance tests are executed on the system. Ideally, this step should take place in the actual environment where the system will be used, but this may be disruptive and impractical. Therefore, a user testing environment may have to be set up to run these tests. It is difficult to automate this process as part of the acceptance tests may involve testing the interactions between end-users and the system. Some training of end-users may be required.

5. Negotiate test results It is very unlikely that all of the defined acceptance tests will pass and that there will be no problems with the system. If this is the case, then acceptance testing is complete and the system can be handed over. More commonly, some problems will be discovered. In such cases, the developer and the customer have to negotiate to decide if the system is good enough to be used. They must also agree on how the developer will fix the identified problems.

6. Reject/accept system This stage involves a meeting between the developers and the customer to decide on whether or not the system should be accepted. If the system is not good enough for use, then further development is required to fix the identified problems. Once complete, the acceptance testing phase is repeated.

What is SapFix ?
Bugs are found via automated testing, SapFix tries to fix the bug automatically using AI to search for similar fixes.

■ Testing can only show the presence of errors in a program. It cannot show that there are no remaining faults.

■ Development testing is the responsibility of the software development team. A separate team should be responsible for testing a system before it is released to customers. In the user testing process, customers or system users provide test data and check that tests are successful.

■ Development testing includes unit testing in which you test individual objects and methods; component testing in which you test related groups of objects; and system testing in which you test partial or complete systems.

■ When testing software, you should try to "break" the software by using experience and guidelines to choose types of test cases that have been effective in discovering defects in other systems.

■ Wherever possible, you should write automated tests. The tests are embedded in a program that can be run every time a change is made to a system.

■ Test-first development is an approach to development whereby tests are written before the code to be tested. Small code changes are made, and the code is refactored until all tests execute successfully.

■ Scenario testing is useful because it replicates the practical use of the system. It involves inventing a typical usage scenario and using this to derive test cases.

■ Acceptance testing is a user testing process in which the aim is to decide if the software is good enough to be deployed and used in its planned operational environment.

CHAPTER 9

Reasons for Software Change
- New requirements emerge when the software is used

- The business environment changes

- Errors must be repaired

- New computers and equipment is added to the system

- The performance or reliability of the system may have to be improved

Software Timeline (Life Cycle)
1. Software Development

2. Software Evolution

3. Software Servicing

4. Software Retirement (Phase-out)

Software Evolution
The stage in a software system's life cycle where it is in operational use and is evolving as new requirements are proposed and implemented in the system.

Software Servicing
At this stage, the software remains useful but the only changes made are those required to keep it operational i.e. bug fixes and changes to reflect changes in the software's environment. No new functionality is added.

Software Retirement (Phase-out)
The software may still be used but no further changes are made to it.

Software Evolution Processes depend on:

- The type of software being maintained

- The development processes used

- The skills and experience of the people involved

General Steps in the Software Evolution Process
1. Change Requests

2. Impact Analysis

3. Release Planning (Includes fault repair, platform adaptation, and system enhancement)

4. Change Implementation

5. System Release

Change Implementation
Iteration of the development process where the revisions to the system are designed, implemented and tested.

General Steps in Change Implementation
1. Proposed changes

2. Requirements analysis

3. Requirements updating

4. Software development

Reasons for urgent change requests
- If a serious system fault has to be repaired to allow normal operation to continue

- If changes to the system's environment (e.g. an OS upgrade) have unexpected effects

- If there are business changes that require a very rapid response (e.g. the release of a competing product).

Handover problems when software is handed over to another development team

- Where the development team have used an agile approach but the evolution team is unfamiliar with agile methods and prefer a plan-based approach. The evolution team may expect detailed documentation to support evolution and this is not produced in agile processes.

- Where a plan-based approach has been used for development but the evolution team prefer to use agile methods. The evolution team may have to start from scratch developing automated tests and the code in the system may not have been refactored and simplified as is expected in agile development.

Legacy Systems

- Legacy systems are older systems that rely on languages and technology that are no longer used for new systems development.

- Legacy software may be dependent on older hardware, such as mainframe computers and may have associated legacy processes and procedures.

- Legacy systems are not just software systems but are broader socio-technical systems that include hardware, software, libraries and other supporting software and business processes.

Components of Legacy Systems

- System hardware: Legacy systems may have been written for hardware that is no longer available.

- Support software: The legacy system may rely on a range of support software, which may be obsolete or unsupported.

- Application software: The application system that provides the business services is usually made up of a number of application programs.

- Application data: These are data that are processed by the application system. They may be inconsistent, duplicated or held in different databases.

- Business processes: These are processes that are used in the business to achieve some business objective.

- Business policies and rules: These are definitions of how the business should be carried out and constraints on the business. Use of the legacy application system may be embedded in these policies and rules.

Legacy System Replacement Risks

- Lack of complete system specification

- Tight integration of system and business processes

- Undocumented business rules embedded in the legacy system

- New software development may be late and/or over budget

Legacy System Change Costs (Why so expensive?)
- No consistent programming style

- Use of obsolete programming languages with few people available with these language skills

- Inadequate system documentation

- System structure degradation

- Program optimizations may make them hard to understand

- Data errors, duplication and inconsistency

Legacy System Management Strategic Options
- Scrap the system completely and modify business processes so that it is no longer required

- Continue maintaining the system

- Transform the system by re-engineering to improve its maintainability

- Replace the system with a new system

Legacy System Assessment Criteria
Business Value vs. System Quality

- High Business Value but Low Quality (make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available.)

- Low Business Value and Low Quality (should be scraped)

- Low Business Value but High Quality (replace with COTS, scrap completely or maintain.)

- High Business Value and High Quality (continue in operation using normal system maintenance)

System Quality Assessment Components
- Business process assessment: How well does the business process support the current goals of the business?

- Environment assessment: How effective is the system's environment and how expensive is it to maintain?

- Application assessment: What is the quality of the application software system?

How should the business value of a legacy system be assessed?
Interview different stakeholders and collate the results!

- System end-users
- Business customers
- Line managers
- IT managers
- senior managers

Issues in Business Value Assessment
- The use of the system: If systems are only used occasionally or by a small number of people, they may have a low business value.

- The business processes that are supported: A system may have a low business value if it forces the use of inefficient business processes.

- System dependability: If a system is not dependable and the problems directly affect business customers, the system has a low business value.

The system outputs: If the business depends on system outputs, then the system has a high business value

Factors used in environment assessment
- Supplier Stability

- Failure Rate

- Age

- Performance

- Support Requirements

- Maintenance Costs

- Interoperability

Factors used in application assessment
- Understandability

- Documentation

- Data

- Performance

- Programming Language

- Configuration Management

- Test Data

- Personnel Skills

System Measurement (Quantitative data collected to make an assessment of the quality of the application system)
- The number of system change requests: The higher this accumulated value, the lower the quality of the system.

- The number of different user interfaces used by the system: The more interfaces, the more likely it is that there will be inconsistencies and redundancies in these interfaces.

- The volume of data used by the system: As the volume of data (number of files, size of database, etc.) processed by the system increases, so too do the inconsistencies and errors in that data. Cleaning up old data is a very expensive and time-consuming process

Software Maintenance
- Modifying a program after it has been put into use.

- The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions.

- Maintenance does not normally involve major changes to the system's architecture.

- Changes are implemented by modifying existing components and adding new components to the system.

Types of Maintenance
- Fault repairs: Changing a system to fix bugs/vulnerabilities and correct deficiencies in the way meets its requirements.

- Environmental adaptation: Maintenance to adapt software to a different operating environment or changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.

- Functionality addition and modification: Modifying the system to satisfy new requirements.

Maintenance Effort Distribution
- 25% Fault Repair

- 20% Environmental Adaptation

- 55% Functionality Addition and Modification

(I adapted these numbers from the book as the book's numbers on the graph represented 101 %....)

Maintenance Costs
- Usually greater than development costs (2x to 100x depending on the application).

- Affected by both technical and non-technical factors.

- Increases as software is maintained. Maintenance corrupts the software structure so makes further maintenance more difficult.

- Ageing software can have high support costs (e.g. old languages, compilers etc.).

Maintenance Costs: Why is it more expensive to add new features during maintenance than it is to add the same features during development?
- A new team has to understand the programs being maintained

- Separating maintenance and development means there is no incentive for the development team to write maintainable software

- Program maintenance work is unpopular

- Maintenance staff are often inexperienced and have limited domain knowledge.

- As programs age, their structure degrades and they become harder to change

Maintenance Prediction
Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs

Change Prediction
- Predicting the number of changes requires and understanding of the relationships between a system and its environment.

- Tightly coupled systems require changes whenever the environment is changed.

- Factors influencing this relationship are:

1) Number and complexity of system interfaces

2) Number of inherently volatile system requirements

3) the business processes where the system is used.

Complexity Metrics
- Predictions of maintainability can be made by assessing the complexity of system components.

- Studies have shown that most maintenance effort is spent on a relatively small number of system components.

- Complexity depends on:

1) Complexity of control structures

2) Complexity of data structures;

3) Object, method (procedure) and module size.

Process Metrics
- Process metrics may be used to assess maintainability:

1) Number of requests for corrective maintenance

2) Average time required for impact analysis

3) Average time taken to implement a change
request

4) Number of outstanding change requests

- If any or all of these is increasing, this may indicate a decline in maintainability.
Software Reengineering
- Restructuring or rewriting part or all of a legacy system without changing its functionality.

- Applicable where some but not all sub-systems of a larger system require frequent maintenance.

- Reengineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented.

Advantages of Reengineering
- Reduced risk: There is a high risk in new software development. There may be development problems, staffing problems and specification problems.

- Reduced cost: The cost of re-engineering is often significantly less than the costs of developing new software.

Reengineering Process Activities
- Source code translation: Convert code to a new language

- Reverse engineering: Analyze the program to understand it

- Program structure improvement: Restructure automatically for understandability

- Program modularization: Reorganize the program structure

- Data reengineering: Clean-up and restructure system data.

Reengineering cost factors
- The quality of the software to be reengineered.

- The tool support available for reengineering.

- The extent of the data conversion which is required.

- The availability of expert staff for reengineering (This can be a problem with old systems based on technology that is no longer widely used)

Refactoring
- Refactoring is the process of making improvements to a program to slow down degradation through change.

- You can think of refactoring as 'preventative maintenance' that reduces the problems of future change.

- Refactoring involves modifying a program to improve its structure, reduce its complexity or make it easier to understand.

- When you refactor a program, you should not add functionality but rather concentrate on program improvement.

Refactoring vs. Reengineering
- Re-engineering takes place after a system has been maintained for some time and maintenance costs are increasing. You use automated tools to process and re-engineer a legacy system to create a new system that is more maintainable.

- Refactoring is a continuous process of improvement throughout the development and evolution process. It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.

"Bad Smells" in Code that can be fixed with Refactoring

- Duplicate code: The same or very similar code may be included at different places in a program. This can be removed and implemented as a single method or function that is called as required.

- Long methods: If a method is too long, it should be redesigned as a number of shorter methods.

- Switch (case) statements: These often involve duplication, where the switch depends on the type of a value. The switch statements may be scattered around a program. In object-oriented languages, you can often use polymorphism to achieve the same thing.

- Data clumping: Data clumps occur when the same group of data items (fields in classes, parameters in methods) re-occur in several places in a program. These can often be replaced with an object that encapsulates all of the data.

- Speculative generality: This occurs when developers include generality in a program in case it is required in the future. This can often simply be removed.

Key Points

- Software development and evolution can be thought of as an integrated, iterative process that can be represented using a spiral model.

- For custom systems, the costs of software maintenance usually exceed the software development costs.

- The process of software evolution is driven by requests for changes and includes change impact analysis, release planning and change implementation.

- Legacy systems are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business.

- It is often cheaper and less risky to maintain a legacy system than to develop a replacement system using modern technology.

- The business value of a legacy system and the quality of the application should be assessed to help decide if a system should be replaced, transformed or maintained.

- There are 3 types of software maintenance, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.

CHAPTER 10

Dependability.
Was proposed to cover the related systems attributes of availability, reliability, safety, and security.

Security Engineering
Designing secure systems that can withstand and recover from external attacks

Elements of: Automated Testing
- Setup
- Call
- Assertion

Reasons for Software Change
- New requirements emerge when the software is used

- The business environment changes

- Errors must be repaired

- New computers and equipment is added to the system

- The performance or reliability of the system may have to be improved

Elements of: Development Testing
- Unit
- Component
- System

Software Timeline (Life Cycle)
1. Software Development

2. Software Evolution

3. Software Servicing

4. Software Retirement (Phase-out)

Confidentiality
Information in a system may only be disclosed or made accessible to people or programs that are authorized to have access to that information. ex: credit card data theft

Hardware failure.
May fail because of mistakes in its design, because components fail as a result of manufacturing errors, or because of environmental factors such as dampness or high temperature.

System failure.
May fail because of mistakes in its specification, design, or implementation.

Elements of: Release Testing
- Requirements
- Scenario
- Performance

Software Evolution
The stage in a software system's life cycle where it is in operational use and is evolving as new requirements are proposed and implemented in the system.

Integrity
Information in a system may be damaged or corrupted, making it unusual or unreliable. ex: worm that deletes data

Operational failure.
Human users may fail to use or operate the system intended by its designers. As hardware and software become more reliable, failures in operation are now, perhaps, the largest cause of system failures.

Elements of: Test Driven Development
- Identify
- Write
- Run

- Implement
- Repeat

Software Servicing
At this stage, the software remains useful but the only changes made are those required to keep it operational i.e. bug fixes and changes to reflect changes in the software's environment. No new functionality is added.

Availability
When normal access to a system or its data is prevented. ex: denial-of-service attack
Infrastructure Security
Concerned with maintaining the security of all systems and networks that provide an infrastructure and a set of shared services to the organization

Software Retirement (Phase-out)
The software may still be used but no further changes are made to it.

Elements of: User Testing
- Alpha
- Beta
- Acceptance

Reliability.
Is the probability, over a given period of time, that the system will correctly deliver services as expected by the user.

Verification & Validation
- Verification: "Are we building the product right?"

- Validation: "Are we building the right product?"

Software Evolution Processes depend on:
- The type of software being maintained

- The development processes used

- The skills and experience of the people involved

Application Security
Concerned with the security of individual application systems or related groups of systems
Safety.
Is a judgment of how likely it is that the system will cause damage to people or its environment?

Development Testing
- AKA Defect Testing, the system is tested to discover bugs and defects

- Includes all testing activities that are carried out by the team developing the system

- Testing activities include Unit testing, Component testing, and System testing

General Steps in the Software Evolution Process
1. Change Requests

2. Impact Analysis

3. Release Planning (Includes fault repair, platform adaptation, and system enhancement)

4. Change Implementation

5. System Release

Operational Security
Concerned with the secure operation and use of the organization's systems

Asset
Something of value that has to be protected. May be the software system itself or the data used by that system?
Ex: patient record

Change Implementation
Iteration of the development process where the revisions to the system are designed, implemented and tested.

Unit Testing
- Stage of Development testing

- Process of testing individual components in isolation.

- It is a defect testing process

- Components to be tested could be individual functions or methods, object classes, or composite components with defined interfaces used to access their functionality.

Security.
Is a judgment of how likely it is that the system can resist accidental or deliberate intrusions?

Resilience.
Is a judgment of how well that system can maintain the continuity of its critical services in the presence of disruptive events, such as equipment failure and cyber-attacks?

General Steps in Change Implementation
1. Proposed changes

2. Requirements analysis

3. Requirements updating

4. Software development

Attack
An exploitation of a system's vulnerability where goal is to cause damage to a system asset or assets. May be external or insider.
Ex: impersonation of an authorized user

Partition Testing
- Test case selection strategy for Unit testing

- Identify groups of inputs that have common characteristics and should be processed in the same way.

- Test cases are chosen from these groups

Reparability.
It must be possible to diagnose the problem, access the component that has failed, and make changes to fix that component.

Guideline-based Testing
- Test case selection strategy for Unit testing

- A set of rules are used to choose test cases

- These rules reflect previous experience of the kinds or errors that programmers often make when developing components.

Reasons for urgent change requests
- If a serious system fault has to be repaired to allow normal operation to continue

- If changes to the system's environment (e.g. an OS upgrade) have unexpected effects

- If there are business changes that require a very rapid response (e.g. the release of a competing product).

Control
A protective measure the reduces a system's vulnerability.
Ex: encryption
ex: password checking system that disallows user password that are proper names or words that are normally included in a dictionary

Handover problems when software is handed over to another development team
- Where the development team have used an agile approach but the evolution team is unfamiliar with agile methods and prefer a plan-based approach. The evolution team may expect detailed documentation to support evolution and this is not produced in agile processes.

- Where a plan-based approach has been used for development but the evolution team prefer to use agile methods. The evolution team may have to start from scratch developing automated tests and the code in the system may not have been refactored and simplified as is expected in agile development.

Exposure
Possible loss or harm to system. Can be loss or damage to data or loss of time/effort due to recovery
ex: loss or reputation
ex: loss of future customers due to lack of trust

Automated Testing
- Recommended wherever possible for Unit Testing

- Making use of a test automation framework (such as Unit) to write and run program tests.

- These frameworks provide generic test cases that you extend to create specific test cases.

Maintainability.
Is software that can be adapted economically to cope with new requirements, and where there is a low probability that making changes will introduce new errors into the system?

Component Interface Testing
- Stage of Development testing

- Software components are often composite components that are made up of several interacting objects accessed through defined component interfaces.

- This type of testing should then focus on showing that the interface behaves according to its specification.

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces

Legacy Systems
- Legacy systems are older systems that rely on languages and technology that are no longer used for new systems development.

- Legacy software may be dependent on older hardware, such as mainframe computers and may have associated legacy processes and procedures.

- Legacy systems are not just software systems but are broader socio-technical systems that include hardware, software, libraries and other supporting software and business processes.

Threat
Circumstances that have potential to cause loss or harm. Basically a system vulnerability that is subjected to an attack
ex: unauthorized user will gain access to the system by guessing the credentials of an authorized user

Error tolerance.
This property can be considered as part of usability and reflects the extent to which the system has been designed, so that user input errors are avoided and tolerated.

Sociotechnical systems.
They include nontechnical elements such as people, processes, and regulations, as well as technical components such as computers, software, and other equipment.

System Testing
- Stage of Development testing

- Involves integrating components to create a version of the system and then testing the integrating system

- Focus is testing the interactions between components

- Checks to ensure that components are compatible, interact correctly, and transfer the right data at the right time.

- Might include stress testing

Components of Legacy Systems
- System hardware: Legacy systems may have been written for hardware that is no longer available.

- Support software: The legacy system may rely on a range of support software, which may be

obsolete or unsupported.

- Application software: The application system that provides the business services is usually made up of a number of application programs.

- Application data: These are data that are processed by the application system. They may be inconsistent, duplicated or held in different databases.

- Business processes: These are processes that are used in the business to achieve some business objective.

- Business policies and rules: These are definitions of how the business should be carried out and constraints on the business. Use of the legacy application system may be embedded in these policies and rules.

Vulnerability
A weakness in a computer-based system that may be exploited to cause loss or harm
ex: authentication is based on a password system that does not require strong passwords

The equipment layer.
Is composed of hardware devices, some of which may be computers.

Test Driven Development (TTD)
- Approach to program development in which you inter-leave testing and code development

- Tests are written before coding and 'passing' the tests is the critical driver of development

- Introduced as a part of agile development methods like extreme programming but can also be employed in plan-driven development processes.

Legacy System Replacement Risks
- Lack of complete system specification

- Tight integration of system and business processes

- Undocumented business rules embedded in the legacy system

- New software development may be late and/or over budget

Security Policies
Rules set in place by a company to ensure the security of a system. For risk management to be effective, organizations should have a documented information security policy that details:
1. the assists that must be protected
2. The level of protection that is required for different types of assets
3. The responsibilities of individual users, managers, and the organization
4. Existing security procedures and technologies that should be maintained

Security Requirements
10 types of security requirements that may be included in a system specification:
1. Identification Requirements - should a system identify its users before interacting with them?
2. Authentication Requirements - how are users identified?
3. Authorization Requirements - what do identified users have access to?
4. Immunity Requirements - how should a system protect itself against viruses, worms, and similar threats
5? Integrity Requirements - how data corruption can be avoided
6. Intrusion Detection - what mechanisms should be used to detect attacks on the system
7? Nonrepudiation Requirements - a party in a transaction cannot deny its involvement in that transaction
8. Privacy Requirements - how data privacy is to be maintained
9. Security Auditing Requirements - how system use can be audited and checked
10. System Maintenance Security Requirements - how an application can prevent authorized changes from accidentally defeating its security mechanisms

The operating system layer.
Interacts with the hardware and provides a set of common facilities for higher software layers in the system.

Legacy System Change Costs (Why so expensive?)
- No consistent programming style

- Use of obsolete programming languages with few people available with these language skills

- Inadequate system documentation

- System structure degradation

- Program optimizations may make them hard to understand

- Data errors, duplication and inconsistency

Regression Testing
- Benefit/Part of Test Driven Development

- A test suite is developed incrementally as a program is developed.

- Checking to ensure that changes to the code have not broken the previously working code.

The communication and data management layer.
Extends the operating system facilities and provides an interface that allows interaction with more extensive functionality, such as access to remote systems and access to a system database.

Release Testing
- Process of testing a particular release of a system that is intended for use outside of the development team

- A separate testing team tests a complete version of the system

- Primary goal is to convince the supplier of the system that it is good enough for use.

- Usually a black-box testing process where tests are only derived from the system specification.

Legacy System Management Strategic Options

- Scrap the system completely and modify business processes so that it is no longer required

- Continue maintaining the system

- Transform the system by re-engineering to improve its maintainability

- Replace the system with a new system

Design guidelines
1. Base security decisions on an explicit security
policy
2. Use defense in depth
3. Fail securely
4. Balance security and stability
5. Log user actions
6. Use redundancy and diversity to reduce risk
7. Specify the format of system inputs
8. Compartmentalize your assets
9. Design for deployment
10. Design for recovery

The application layer.
Delivers the application-specific functionality that is required.

Requirements Based Testing
- Part of Release Testing

- Involves examining each requirement and developing a test or tests for it.

- Testing what was written up for and documented for the system.

- It is validation rather than defect testing: you are trying to demonstrate that the system has properly implemented its requirements.

Legacy System Assessment Criteria
Business Value vs. System Quality

- High Business Value but Low Quality (make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available.)

- Low Business Value and Low Quality (should be scraped)

- Low Business Value but High Quality (replace with COTS, scrap completely or maintain.)

- High Business Value and High Quality (continue in operation using normal system maintenance)

The business process layer.
Includes the organizational business processes, which make use of the software system.

Scenario Testing
- Part of release testing.

- An approach to release testing whereby you devise typical scenarios of use and use these scenarios to develop test cases for the system.

System Quality Assessment Components
- Business process assessment: How well does the business process support the current goals of the business?

- Environment assessment: How effective is the system's environment and how expensive is it to maintain?

- Application assessment: What is the quality of the application software system?

How should the business value of a legacy system be assessed?
Interview different stakeholders and collate the results!

- System end-users
- Business customers
- Line managers
- IT managers
- senior managers

Performance Testing
- Part of release testing

- AKA Stress Testing

- Concerned with both demonstrating that the system meets its requirements and discovering problems and defects within the system.

- You are testing the system by making demands that are outside the design limitations.

The organizational layer.
Includes higher-level strategic processes as well as business rules, policies, and norms that should be followed when using the system.

The social layer.
Refers to the laws and regulations of society that govern the operation of the system.

User Testing
- Potential users of a system operate the system in their own environment.
Issues in Business Value Assessment
- The use of the system: If systems are only used occasionally or by a small number of people, they may have a low business value.

- The business processes that are supported: A system may have a low business value if it forces the use of inefficient business processes.

- System dependability: If a system is not dependable and the problems directly affect business customers, the system has a low business value.

The system outputs: If the business depends on system outputs, then the system has a high business value

Regulation and compliance.
Applies to the sociotechnical system as a whole and not simply the software element of that system.

Alpha Testing
- Stage of user testing

- Users of the software work with the development team to test the software at the developer's site.

Factors used in environment assessment
- Supplier Stability

- Failure Rate

- Age

- Performance

- Support Requirements

- Maintenance Costs

- Interoperability

Redundancy.
Means that spare capacity is included in a system that can be used if part of that system fails.

Beta Testing
- Stage of user testing

- A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.

Factors used in application assessment
- Understandability

- Documentation

- Data

- Performance

- Programming Language

- Configuration Management

- Test Data

- Personnel Skills

Diversity.
Means that redundant components of the system are of different types, thus increasing the chances that they will not fail in exactly the same way.

Acceptance Testing
- Stage of user testing

- Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

System Measurement (Quantitative data collected to make an assessment of the quality of the application system)
- The number of system change requests: The higher this accumulated value, the lower the quality of the system.

- The number of different user interfaces used by the system: The more interfaces, the more likely it is that there will be inconsistencies and redundancies in these interfaces.

- The volume of data used by the system: As the volume of data (number of files, size of database, etc.) processed by the system increases, so too do the inconsistencies and errors in that data. Cleaning up old data is a very expensive and time-consuming process

Dependable software processes.
Are software processes that are designed to produce dependable software? They make use of redundancy and diversity to achieve reliability.

Software Maintenance
- Modifying a program after it has been put into use.

- The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions.

- Maintenance does not normally involve major changes to the system's architecture.

- Changes are implemented by modifying existing components and adding new components to the system.

Requirements reviews.
To check that the requirements are, as far as possible, complete and consistent.

Types of Maintenance
- Fault repairs: Changing a system to fix bugs/vulnerabilities and correct deficiencies in the way meets its requirements.

- Environmental adaptation: Maintenance to adapt software to a different operating environment or changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.

- Functionality addition and modification: Modifying the system to satisfy new requirements.

Requirements management.
To ensure that changes to the requirements are controlled and that the impact of proposed requirements changes is understood by all developers affected by the change.

Maintenance Effort Distribution
- 25% Fault Repair

- 20% Environmental Adaptation

- 55% Functionality Addition and Modification

(I adapted these numbers from the book as the book's numbers on the graph represented 101 %....)

Formal specification.
Where a mathematical model of the software is created and analyzed.

Maintenance Costs
- Usually greater than development costs (2x to 100x depending on the application).

- Affected by both technical and non-technical factors.

- Increases as software is maintained. Maintenance corrupts the software structure so makes further maintenance more difficult.

- Ageing software can have high support costs (e.g. old languages, compilers etc.).

System modeling.
Where the software design is explicitly documented as a set of graphical models and links between the requirements and these models are explicitly documented.

Maintenance Costs: Why is it more expensive to add new features during maintenance than it is to add the same features during development?
- A new team has to understand the programs being maintained

- Separating maintenance and development means there is no incentive for the development team to write maintainable software

- Program maintenance work is unpopular

- Maintenance staff are often inexperienced and have limited domain knowledge.

- As programs age, their structure degrades and they become harder to change

Design and program inspections.
Where the different descriptions of the system are inspected and checked by different people.

Maintenance Prediction
Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs

Static analysis.
Where automated checks are carries out on the source code of the program.

Change Prediction
- Predicting the number of changes requires and understanding of the relationships between a system and its environment.

- Tightly coupled systems require changes whenever the environment is changed.

- Factors influencing this relationship are:

1) Number and complexity of system interfaces

2) Number of inherently volatile system requirements

3) the business processes where the system is used.

Test planning and management.
Where a comprehensive set of system tests is designed.

Complexity Metrics
- Predictions of maintainability can be made by assessing the complexity of system components.

- Studies have shown that most maintenance effort is spent on a relatively small number of system components.

- Complexity depends on:

1) Complexity of control structures

2) Complexity of data structures;

3) Object, method (procedure) and module size.
Specification and design errors and omissions.

The process of developing and analyzing a formal model of the software may reveal errors and omissions in the software requirements.
Process Metrics
- Process metrics may be used to assess maintainability:

1) Number of requests for corrective maintenance

2) Average time required for impact analysis

3) Average time taken to implement a change request

4) Number of outstanding change requests

- If any or all of these is increasing, this may indicate a decline in maintainability.

Inconsistencies between a specification and a program.

If a refinement method is used, mistakes made by developers that make the software inconsistent with the specification are avoided.

Software Reengineering
- Restructuring or rewriting part or all of a legacy system without changing its functionality.

- Applicable where some but not all sub-systems of a larger system require frequent maintenance.

- Reengineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented.

Advantages of Reengineering
- Reduced risk: There is a high risk in new software development. There may be development problems, staffing problems and specification problems.

- Reduced cost: The cost of re-engineering is often significantly less than the costs of developing new software.

Reengineering Process Activities
- Source code translation: Convert code to a new language

- Reverse engineering: Analyze the program to understand it

- Program structure improvement: Restructure automatically for understandability

- Program modularization: Reorganize the program structure

- Data reengineering: Clean-up and restructure system data.

Reengineering cost factors
- The quality of the software to be reengineered.

- The tool support available for reengineering.

- The extent of the data conversion which is required.

- The availability of expert staff for reengineering (This can be a problem with old systems based on technology that is no longer widely used)

Refactoring
- Refactoring is the process of making improvements to a program to slow down degradation through change.

- You can think of refactoring as 'preventative maintenance' that reduces the problems of future change.

- Refactoring involves modifying a program to improve its structure, reduce its complexity or make it easier to understand.

- When you refactor a program, you should not add functionality but rather concentrate on program improvement.

Refactoring vs. Reengineering
- Re-engineering takes place after a system has been maintained for some time and maintenance costs are increasing. You use automated tools to process and re-engineer a legacy system to create a new system that is more maintainable.

- Refactoring is a continuous process of improvement throughout the development and evolution process. It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.

"Bad Smells" in Code that can be fixed with Refactoring
- Duplicate code: The same or very similar code may be included at different places in a program. This can be removed and implemented as a single method or function that is called as required.

- Long methods: If a method is too long, it should be redesigned as a number of shorter methods.

- Switch (case) statements: These often involve duplication, where the switch depends on the type of a value. The switch statements may be scattered around a program. In object-oriented languages, you can often use polymorphism to achieve the same thing.

- Data clumping: Data clumps occur when the same group of data items (fields in classes, parameters in methods) re-occur in several places in a program. These can often be replaced with an object that encapsulates all of the data.

- Speculative generality: This occurs when developers include generality in a program in case it is required in the future. This can often simply be removed.

- Software development and evolution can be thought of as an integrated, iterative process that can be represented using a spiral model.

- For custom systems, the costs of software maintenance usually exceed the software development costs.

- The process of software evolution is driven by requests for changes and includes change impact analysis, release planning and change implementation.

- Legacy systems are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business.

- It is often cheaper and less risky to maintain a legacy system than to develop a replacement system using modern technology.

- The business value of a legacy system and the quality of the application should be assessed to help decide if a system should be replaced, transformed or maintained.

- There are 3 types of software maintenance, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.

CHAPTER 11

The build process is a set of activities associated with the integration and conversion of source files to a set of executable files targeted for a specific execution environment.
(True or False)
True

The merge function combines a change in one artifact into another artifact.
(True or False)
True

Software configuration is done at the conclusion of a software project.
(True or False)
False

Software configuration management is the process of managing all the pieces and parts of artifacts produced as part of software development and support activities.
(True or False)
True

Given a software artifact that we maintain in four different country versions and two platforms (IOS and Android), with yearly releases for 5 years, how many versions of the artifact would need to be managed?

Answers:
a. 40
b. 8
c. 11
d. 60

a.) 40

The two major steps involved in the build cycle for a single program are:

Answers:
a. design and implementation.
b. compile and link.
c. implement and test.
d. analysis and design.
b.) Compile and Link

In order to ensure that all the source materials are the right ones during and throughout the development stages, we must introduce and implement the discipline of configuration management.
(True or False)
True

_____ is the process of managing all the pieces and parts of artifacts produced as part of software development and support activities.
management

Answers:
a. Configuration management framework
b. Software configuration management
c. Software engineering
d. Build process
b.) Software configuration management

Which of the following models defines the facilities needed to store and to control the access of all the software artifacts?

Answers:
a. Build
b. Storage and access
c. Software configuration management
d. Naming

b.) Storage and access

In order to ensure that all the source materials are the right ones during and throughout the development stages, we must introduce and implement the discipline of configuration management.
(True or False)
True

The fundamental discipline of software configuration management involves keeping clear account of the multiple versions of source material and being able to deliver any one of those versions for integration and build, which generates the desired software release for users.
(True or False)
True

Companies with heavier, waterfall processes will tend to maintain more artifacts and more versions of each artifact, and companies with lighter, agile processes will tend to keep only the

latest version.
(True or False)
True

To properly control all the parts, there must be a way to uniquely identify each artifact.
(True or False)
True

The requirements artifacts are usually tied by our build tools to the related design document, source code, and test cases.
(True or False)
False

Which of the following is a popular build tool from the C and UNIX system days of the 1970s and 1980s?

Answers:
a. SourceSafe
b. Make
c. Gradle
d. Git

b.) Make

The fundamental discipline of which of the following involves keeping clear account of the multiple versions of source material and being able to deliver any one of those versions for integration and build, which generates the desired software release for users?

Answers:
a. Software engineering
b. Software configuration management
c. Software development
d. Building tools

b.) Software configuration management

For large, complex software, the complete build cycle may take several hours.
(True or False)
True

Which of the following tools are used for version control?

Answers:
a. Make
b. Git
c. Gradle
d. None of these are correct.

b.) Git

Which one of the following activities is not one of the components of software configuration management?

Answers:
a. Training and ensuring that the agreed-upon configuration management process is practiced and adhered to

b. Determining and defining the framework that needs to be used to manage these artifacts

c. Managing the software development life cycle process

d. Understanding the policy, process activities, and the resulting artifacts that need to be managed

c.) Managing the software development life cycle process

Which of the following service functions allows the release number or the version number to be explicitly incremented by the desired amount?

Answers:
a. Gather
b. Add
c. Merge
d. Increment

CHAPTER 15

Software Reuse
- Systems are designed by composing existing components that have been used in other systems.

- There has been a major switch to reuse-based development over the past 10 years.

Levels of Software Reuse
1) System Reuse: Complete systems, which may include several application programs may be reused.

2) Application Reuse: An application may be reused either by incorporating it without change into other or by developing application families.

3) Component Reuse: Components of an application from sub-systems to single objects may be reused.

4) Object and Function Reuse: Small-scale software components that implement a single well-defined object or function may be reused.

Benefits of Software Reuse
- Accelerated development

- Effective use of specialists

- Increased dependability

- Lower Development costs

- Reduced process risk

- Standards compliance

Problems with Software Reuse
- Creating, Maintaining and using a component library

- Finding, understanding and adapting reusable components

- Increased maintenance costs

- Lack of tool support

- Not-invented-here syndrome (Also means lack of documentation)

Approaches that Support Software Reuse
- Application frameworks

- Application System Integration

- Architectural Patterns

- Aspect-Oriented Software Development

- Component-based Software Development

- Configurable Application Systems

- Design Patterns

- ERP Systems

- Legacy System Wrapping

- Model-Driven Engineering

- Program Generators

- Program Libraries

- Service-Orientated Systems

- Software Product Lines

- Systems of Systems

Key factors when Determining to Reuse Software
1) Development schedule for the software

2) The expected software lifetime

3) The background, skills, and experience of the development team

4) The criticality of the software and its non-functional requirements

5) The application domain

Definition of a Framework
An integrated set of software artifacts (such as classes, objects and components) that collaborate to provide a reusable architecture for a family of related applications.

Application Frameworks
- Moderately large entities that can be reused

- Sub-system design made up of a collection of abstract and concrete classes and the interfaces between them.

- Sub-system is implemented by adding components to fill in parts of the design and by instantiating the abstract classes in the framework.

Model-View-Controller
- System infrastructure framework for GUI design.

- Allows for multiple presentations of an object and separate interactions with these presentations.

- MVC framework involves the instantiation of a number of patterns

Web Application Frameworks
- Support the construction of dynamic websites as a front-end for web applications.

- WAFs are now available for all of the commonly used web programming languages e.g. Java, Python, Ruby, etc.

- Interaction model is based on the Model-View-Controller composite pattern.
WAF Features
- Security: WAFs may include classes to help implement user authentication (login) and access.

- Dynamic web pages: Classes are provided to help you define web page templates and to populate these dynamically from the system database.

- Database support: The framework may provide classes that provide an abstract interface to different databases.

- Session management: Classes to create and manage sessions (a number of interactions with the system by a user) are usually part of a WAF.

- User interaction: Most web frameworks now provide AJAX support, which allows more interactive web pages to be created.

Framework Classes
- System infrastructure frameworks: Support the development of system infrastructures such as communications, user interfaces and compilers.

- Middleware integration frameworks: Standards and classes that support component communication and information exchange.

- Enterprise application frameworks: Support the development of specific types of application such as telecommunications or financial systems.

Software Product Lines
- Software product lines or application families are applications with generic functionality that can be adapted and configured for use in a specific context.

- Set of applications with a common architecture and shared components

Product Line Specialization
- Platform specialization: Different versions for different platforms

- Environment specialization: Different versions created to handle different environments

- Functional specialization: Different versions created for different requirements

- Process specialization: Different versions created to support different business processes

Process for Extending a Software Product Line
1) Elicit stakeholder requirements: Use existing family member as prototype

2) Choose closest-fit family member: Member best meets the requirements

3) Re-negotiate requirements: Adapt requirements as necessary

4) Adapt existing system

5) Deliver new family member

Product Line Configuration
1) Design time configuration: Modifies a common product to create a new system for customer

2) Deployment time configuration: Generic system designed for configuration by a customer

Levels of Deployment Time Configuration
1) Component selection: where you select the modules in a system that provide the required functionality.

2) Workflow and rule definition: where you define workflows (how information is processed, stage-by-stage) and validation rules that should apply to information entered by users or generated by the system.

3) Parameter definition: where you specify the values of specific system parameters that reflect the instance of the application that you are creating

Application System Reuse
- Application system product is a software system that can be adapted for different customers without changing the source code of the system

- Application systems have generic features so can be used/reused in different environments. Products adapted by using built in configuration mechanisms.

- Allow functionality of the system to be tailored

Benefits of Application System Reuse
- More rapid deployment

- See what functionality is provided

- Development risks are avoided by using existing software

- Businesses can focus on core activity w/out having to devote IT resources

- Updates can be simplified

Problems of Application System Reuse
- Requirements usually need to be adapted

- COTS product may be based on assumptions

- Choosing the right COTS system for an enterprise can be difficult

- Lack of local expertise to support systems

- COTS product vendor controls system support and evolution

Configurable Application Systems
- Configurable application systems are generic application systems that may be designed to support a particular business type

- Domain-specific systems, such as systems to support a business function (e.g. document management) provide functionality that is likely to be required by a range of potential users.

ERP Systems
- An Enterprise Resource Planning (ERP) system is a generic system that supports common business processes such as ordering and invoicing, manufacturing, etc.

- These are very widely used in large companies - they represent probably the most common form of software reuse.

- The generic core is adapted by including modules and by incorporating knowledge of business processes and rules.

ERP Architecture
- A number of modules to support different business functions.

- A defined set of business processes, associated with each module, which relate to activities in that module.

- A common database that maintains information about all related business functions.

- A set of business rules that apply to all data in the database.

ERP Configuration
1) Selecting the required functionality from the system.

2) Establishing a data model that defines how the organization's data will be structured in the system database.

3) Defining business rules that apply to that data.

4) Defining the expected interactions with external systems.

5) Designing the input forms and the output reports generated by the system.

6) Designing new business processes that conform to the underlying process model supported by the system.

7) Setting parameters that define how the system is deployed on its underlying platform.

Integrated Application Systems
- Applications that include two or more application system products and/or legacy application systems.

- You may use this approach when there is no single application system that meets all of your needs or when you wish to integrate a new application system with systems that you already use.

Service-Oriented Interfaces
- Application system integration can be simplified if a service-oriented approach is used.

- A service-oriented approach means allowing access to the application system's functionality through a standard service interface, with a service for each discrete unit of functionality.

- Some applications may offer a service interface but, sometimes, this service interface has to be implemented by the system integrator. You have to program a wrapper that hides the application and provides externally visible services.

Application system integration problems
- Lack of control over functionality and performance: Application systems may be less effective than they appear

- Problems with application system inter-operability: Different application systems may make different assumptions that means integration is difficult

- No control over system evolution: Application system vendors not system users control evolution

- Support from system vendors: Application system vendors may not offer support over the lifetime of the product

- There are many different ways to reuse software. These range from the reuse of classes and methods in libraries to the reuse of complete application systems.

- The advantages of software reuse are lower costs, faster software development and lower risks. System dependability is increased. Specialists can be used more effectively by concentrating their expertise on the design of reusable components.

- Application frameworks are collections of concrete and abstract objects that are designed for reuse through specialization and the addition of new objects. They usually incorporate good design practice through design patterns.

- Software product lines are related applications that are developed from one or more base applications. A generic system is adapted and specialized to meet specific requirements for functionality, target platform or operational configuration.

- Application system reuse is concerned with the reuse of large-scale, off-the-shelf systems.

These provide a lot of functionality and their reuse can radically reduce costs and development time. Systems may be developed by configuring a single, generic application system or by integrating two or more application systems.

- Potential problems with application system reuse include lack of control over functionality and performance, lack of control over system evolution, the need for support from external vendors and difficulties in ensuring that systems can inter-operate.

CHAPTER 18

Web service
a loosely coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed; accessed using standard internet and XML-based protocols

Service
act or performance offered by one party to another; although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production

Benefits of Service-Oriented approach to Software Engineering (6)
1. Services can be offered by any service provider inside or outside an organization; services from a range of providers may be incorporated into an application
2. Provider makes service information public; any authorized user can use; no negotiation needed
3. Applications can be reactive and adapt their operation to cope with changes to their execution environment
4. Opportunistic construction of new services is possible; provider may link existing services in innovative ways to create new services
5. Users can pay for services according to their use rather than their provision
6. Applications can be made smaller; intensive processing and exception handling can be offloaded to external services

Service oriented software engineering promises to _____ (4)
Significantly improve corporate agility
Speed time-to-market for new products and services
Reduce IT costs
Improve operational efficiency

Initial work on service provision was heavily influenced by _____
failure of software industry to agree on component standards
Service-oriented architecture (SOA)
architectural style based on the idea that executable services can be included in applications
3 Fundamental Standards for Service-Oriented Architecture

1. SOAP; message interchange standard that supports communication between services
2. WSDL; Web Service Description Language; standard service interface definition
3. WS-BEPL; standard for a workflow language that is used to defined process programs involving several different services

UDDI (Universal Description, Discovery, and Integration)
discovery standard that defines the components of a service specification intended to help potential users discover the existence of a service

Message exchange
important mechanism for coordinating actions in a distributed computing system; this is how services in SOA communicate; these ____s are expressed in XML, and are distributed using standard Internet transport protocols such as HTTP and TCP/IP
3 aspects of a web service that WSDL defines
1. the "what" (the interface); what operations the service supports and defines format of messages sent and received by the service
2. the "how" (the binding); maps abstract interface to a concrete set of protocols
3. the "where"; location of specific web service implementation

Elements of a WSDL specification (6)
1. intro part, defines XML namespaces used
2. optional description of types used in messages exchanged by a service
3. description of the service interface (what it provides)
4. description of input/output msgs processed by service
5. description of binding (messaging protocol) used by service; default is SOAP
6. endpoint specification (physical location of service expressed as URI)
REST

Representational State Transfer; architectural style based on transferring representations of resources from a server to a client; style that underlies web as a whole; simpler than SOAP/WSDL for implementing web service interfaces

fundamental element in a RESTful architecture is a _____, an example of which is a ____ which has a _____
resource; web page; URL (unique resource ID)

4 Fundamental Polymorphic Operations associated with resources in a RESTful architectures
1. Create - bring resource into existence
2. Read - return a representation of the resource
3. Update - change the value of the resource
4. Delete - make inaccessible

web protocols http and https are based on what 4 actions?
1. POST; used to create resource
2. GET ; used to read value of resource
3. PUT; used to update value of resource
4. DELETE; used to delete the resource

RESTfurl services should be _____
stateless; resources themselves should not include any state information, such as time of last request

Service engineering
Process of developing services for reuse in service-oriented applications
3 logical stages in service engineering process
1. Service candidate identification
2. Service design
3. Service implementation/deployment

3 fundamental types of service
1. utility
2. business
3. coordination/process

Goal of service candidate identification in service engineering process:
identify services that are logically coherent, independent, and reusable

Starting point for service interface design is ____
abstract interface design; identify entities and operations associated with the service, their inputs/outputs, and the exceptions associated with these operations

service interface design
defining operations associated with the service and their parameters
How should service exception handling be approached?

Service developers should not impose their views on how exceptions should be handled; leave all exception handling to the user of the component

Service testing
involves examining and partitioning service inputs, creating input messages that reflect these input combinations, checking that outputs are expected; try to generate exceptions!

Service deployment
making service available for use on a web server; may need to provide documentation for external service users;

underlying principle of service-oriented software engineering is _____
that you compose and configure services to create new, composite services

Designing with reuse inevitably involves _____
requirements compromises

6 Key stages in process of system construction by composition
1. Formulate outline workflow; "ideal" service design; abstraction
2. Discover services;
3. Select possible services; which ones offer appropriate functionality? cost? QoS?
4. Refine workflow; add detail to abstraction
5. Create workflow program; transform abstract design into executable program, define service interface
6. Test completed service/application

Workflow design
analyzing existing or planned business processes to understand the tasks involved and how these tasks exchange information; then define new business process in a workflow design notation
BPMN
business process modeling notation

Services are _____-independent
implementation language

Problems faced when testing service composition (4)
1. external services are under control of service provider rather than user
2. if services are dynamically bound, an app may not be using same service each time it is executed
3. Non-functional behavior of a service not SOLELY dependent on how it is used by application
4. Payment model for services could make service testing very expensive

When are service composition testing problems more acute?
when external services are used